Scientia Iranica, Vol. 10, No. 4, pp 454–463
© Sharif University of Technology, October 2003

Research Note

Wrapping Data for Querying

A. Fatholahzadeh[1]

Various tools and systems for querying are developed and available for applications. In this paper, breaking away from the conventional querying, a new technique, Q2, is presented in which querying is allowed by user's style sheets via the long strings, which may contain the implicit information. For mining tasks of the query processing, our own wrapper is proposed, composed of two components: 1) A set of Functional Dependency Plots (FDP) of data; 2) A Metabase: A database containing the storage-orders of used data. In order to speed up the query processing, other new ingredients have been incorporated in Q2; 3) A lexicon of terms including technical jargon; 4) A Knowledge Base System (KBS), dealing with declarative and background knowledges for identifying relevant databases with respect to the query at hand; 5) Encoder: For obtaining variable (column) names of the selected databases by KBS; 6) Decoder: For translating some KBS terms via the lexicon in order to be usable by the encoder; 7) Interpolator: A mathematical tool using FDP for outputting the response of the queries. Q2 has been implemented using the C language and NEXPERT tool and has been applied to the interpretation of laser-material experiments.

INTRODUCTION

With more and more databases created, an increasingly pressing issue is how to make efficient use of them. To address the efficiency issue, there has been a recent surge of research interest on knowledge discovery and data mining [1-5]. In [4], an algorithm has been proposed to deal with data aspects in the knowledge discovery process for identifying relevant databases. The authors argue that the first step for multi-database mining is to identify databases that are the most likely to be relevant to an application. However, in [4] the task is to select the databases relevant to a given query predicate rather than identifying the databases matching the query.

In this work, the identification of relevant databases matching the queries expressed in user style sheets will be addressed and it will be argued that using our wrapper contributes to the efficiency aspect of the query processing. The allowed style sheet in this work is one or more templates. A template is just a long string. A long string is a set of keys (e.g. "Renault 21 TD"). A key (noted by k) is referred to as a sequence of characters surrounded by empty spaces but containing no internal space. A typical example of such query could be the following style sheet: "Renault 21 TD" (or "Renault-21-TD") and "5000-6000 Euros"; a self explanatory query (just compare its easiness with a program written in, say, SQL) referring to a particular car with a limited budget, which could be used in commercial transactions.

Although Querying by Long Strings (QLS) has the advantage of being syntax free, this makes the problem of the query processing all the more difficult, since the long strings may contain the implicit information from more general to make more specific. Therefore, when querying by long strings, the mining task is to explicit the implicit information with respect to the query at hand.

Among various data mining tasks (classification, characterization, association, etc.), most of them aim at discovering knowledge that can be expressed as relationships among values of attributes. For example, classification rules, characteristic rules and association rules can be expressed in the form of $A \rightarrow B$, where both A and B are conjunctions of attribute values. Queries can easily take care of these forms of knowledge.

In this work, data mining tasks of the physical systems are of interest, which can be characterized by general knowledge of the form $y = f(x_1, \cdots x_n)$ where

1. *Supélec-Campus de Metz, 2, rue Édouard Belin, 57078 Metz, France.*

x_i denotes a variable. To our knowledge, no work on mining tasks of such systems, with respect to the query processing, has been reported in the literature.

It may happen that several values become associated with the same variable, due to the different conditions of the experimentations and, in particular, these values become recorded into the same database under different columns (names of attributes). Table 1 shows such a database, where three facets of the same variable, namely $\frac{T}{PD}$, are given and where the abbreviations of T and PD stand for the time and the power density (of a Carbon-Resin laser). In this paper, such variable will be called multi-facets-attribute.

Without loss of generality, each database is called a relation or table and it is assumed that people are likely to pose queries using their own style sheets rather than expressing them via a query language (e.g. SQL) and the user is supposed to have partial knowledge (i.e. rough idea) of the category (e.g. car) to which querying is applied. Note that the second assumption is not a requirement in this work, as one has the possibility of using the suggested templates by Q2 to pose the queries to Q2.

In this paper, the problem of the quality in user-controlled querying will not be addressed, since it is similar to that of user-controlled navigation of WWW communities. Recently, in [6], the use of free-syntax long strings is advocated, as the style specifications of this model is new. Quality in user-controlled navigation remains the biggest challenge for search engines [7]. So, efficient processing of long strings helps to improve the quality. This is because the long strings of the query constrain web browsing, search and navigation using the user's own background knowledge. User-controlled navigation has been studied by several researchers from the WWW communities, under the name of spiders, crawlers, robots, knowbots and so on. In [8] a kind of style using C++ is given. The modification of the query language, SQL, to operate on the web, is proposed in [9]. A system which piggy-backs onto a Lycos browser is described in [10].

In this paper, the relevance problem will be addressed: Identifying databases relevant to a particular data mining task in multi-database and matching the query with selected relevant databases. It is important to note that unlike the method described in [4], the technique in this paper takes care of the multi-facets-attributes and regarding conventional query processing, the query predicate is expressed in the form of a template (i.e. long string) rather than using the syntax of a given query language (e.g. SQL).

PRELIMINARIES

The query template is denoted by Q_t, such as "Group is Chinese" and "How long does it take to perforate ...", etc. Let it be supposed that one is equipped with a function, namely, Transform Query (Q_t) outputting the query predicate, Q, such that Q can be expressed in the form of "A relop C", where A is an attribute name, relop is one of the relational operators $\{=, >, <, \leq, \geq, \neq\}$ and C is a value in its domain.

If the attribute A is not referred by the query predicate Q, where C is a constant value in the domain of A, then the form "A relop C" will be called a selector (noted by s).

A long string will be called a selector template and will be denoted by s_t if it can be transformed into a selector. An example of s_t is the form of: "diameter (of the steel) is 2 mm".

Note that unlike in [4], in this work the constant value of an attribute, C, of a selector is not necessarily and explicitly present in one or more databases. If any, the relevant functional dependency plot will be used for outputting the response of the query.

The problem described in this paper can be stated as follows: Given n data tables (relations), $D_K(1 \leq k \leq n)$, each of which consists of a number of attributes, a query template Q_t and one or more selector templates, s_t. Perform the following tasks: Identify relevant tables (relations) that contain specific information pertaining to the query template Q_t and match Q_t with selected relevant databases.

Related Work on Identifying Relevant Databases

There may be an objection as to why not employ a brute force approach to join the available tables into a single large table upon which existing techniques or

Table 1. Carbon-resin database (the data are fictitious).

AF	$\frac{T}{PD}$ **= 950 W/cm²**	$\frac{T}{PD}$ **= 500 W/cm²**	$\frac{T}{PD}$ **= 240 W/cm²**
1066	151.30	81.70	323.00
1035	278.00	150.30	464.00
990	462.00	249.80	220.30
941	669.00	358.10	581.70
837	940.00	508.50	667.00
$\Re_<$	$\Re_>$	$\Re_>$	$\Re_!$(mixed order)

tools can be applied. There are several problems for this approach in real-world applications. Prima facie is the database integration, especially where the source domains differ. Second, all tables with foreign key references need to be joined together to produce a single combined table. The size of the resulting table, in terms of both the number of records and the number of attributes, will be much larger than the original tables. The increase of data size not only penalizes the running time of mining algorithms, but also affects the behavior of mining algorithms.

From the viewpoint of statistics, joining one relevant database with an irrelevant database will result in a more difficult task to find useful patterns, as search space is enlarged by irrelevant attributes. Suppose that there are two binary-valued databases and each has n attributes, assuming that one database is irrelevant and $\frac{n}{2}$ attributes can be found in both databases. Simply working on the relevant database, hypothesis space is 4^n; after joining, it is $4^{\frac{3n}{2}}$. Note that this analysis is a simplified one, because the factor of a missing value due to joining, is not considered. If any, this factor will also certainly increase the difficulty of the task.

Liu et al. [4] developed a method to select the relevant databases. The method is based on a quantitative measure of the relevance of the query Q, which, in turn, is based on the Relevance Factor (RF) of the selector, s. RF of s such as "drink = tea", with respect to Q, such as, "group = Chinese", is defined by a probabilistic method. The mining task of the method depends on choosing a value for the threshold, δ. For example, let it be assumed that in a given database, say diet habit, there are two columns of "group" and "drink". If one thinks that for inferring knowledge such as, "drink = tea $\rightarrow$ group = Chinese" there should be at least 80 % of the data satisfying the premise part of the mentioned rule, then that database is relevant to the query "group = Chinese", with respect to the selector "drink = tea".

WRAPPER

A wrapper is a reservoir of useful information-telephone directories, product catalogs, stock quotes etc., which can be used as the important part of the retrieval information's engine for providing multiple tasks, including querying. Recently, in the WWW communities, many systems have been built that automatically gather and manipulate such information on the user's behalf. In [11] a method of wrapper induction is given. However, to the author's knowledge, the design and the utilization of the wrapper for the query processing appears to be new.

As mentioned earlier, the wrapper described in this paper, is composed of two reservoirs: the first one is the set of the Functional Dependency Plots (FDP) of data. The second reservoir is a special database called metabase. Each FDP relates one or more multi-facet-attributes and describes the dependency between the different facets of two variables of a class/object (e.g. CO_2, Carbon-Resin) belonging to category (e.g. laser) at hand. Furthermore, each FDP is generated using a graphical tool (e.g. gnuplot) whereas the second reservoir is formed manually. Take note of the graphical curves depicted by Figure 1 and suppose an interest in using it for querying purpose, such as the following:

How long does it take to perforate a 2 mm thick sheet of steel with a CO_2 laser irradiation of 3500 W/cm^2?

Regarding the present query, obviously, it is desirable to exploit the dependency between the times of the perforations of a particular polished steel and the intensities of CO_2-laser using the (discrete) data of Figure 1 and, in particular, as much as possible, generalize such an approach. The processing of this task is based on the concept of functional dependency, which was introduced in the theory of relational databases (see [12]) in the seventies, and it has been commonly used in the logical database design ever since (see e.g. [13-15]). In [16], this concept was applied to the field of knowledge representation.

A functional dependency states that in all models of a theory, the value of a variable is a function of values of some other variables. Thus, the existence of a functional dependency is an important property of the theory. If a functional dependency holds in a theory, then it may be possible to simplify the theory by eliminating the variable, whose value is determined by other variables. The repeated applications of this procedure will result in a so-called "condensed" theory, which does not have any functional dependencies and may have much fewer variables than the original theory.

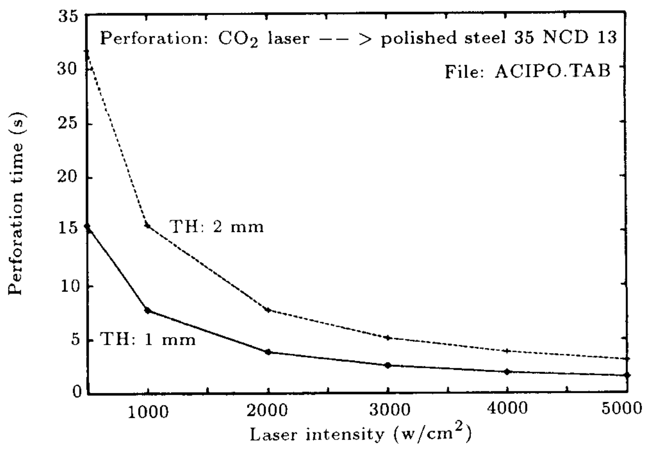

Figure 1. Graphical curves (interpolator) showing the dependency between the times of the perforation of the polished steel and the intensity of CO_2-laser.

Note that, in general, the condensed theory can be harder to reason with.

According to the principle of the condensed theory, which is idealized in the following terms: "use fewer variables than those of the original theory", instead of recording the data associated with every facet of a variable, only the data of one facet was kept for each variable. For instance, instead of having two variables for the thickness of the material in the CO_2 table, having 1 mm and 2 diameters, only the data of 1 mm was retained in the mentioned table. If the query refers to a material with the thickness of 2 mm (i.e. $C = 2$ mm), after having determined the response for $C = 1$ mm via the relevant database, the interpolator of the same database is used to calculate the searched value. The last line of Table 2 gathers the storage order of the data in columns 1 to 4 of the carbon-resin database of Table 1, which are decreasing, increasing and mixing, respectively. Table 2 shows a part of the metabase with respect to the application at hand, namely, laser-material interaction. The first column (field) of the metabase starts always with a symbol, namely either/or # and is followed by the name of a database. The symbol '/' is served to gather the similarity of the storage-orders that may exist between the data of two or more databases. Therefore, for instance, the first line of Table 2 means that the storage-orders of 'acirect' should be searched via the storage-orders of another, namely 'acirpol'. The symbol '#' is used for directly giving the information related to the storage-orders. For instance, the storage-orders of 'acirpol' are shown in the second line of Table 2, where the characters ':' and '-' are used as the separators between the used tokens (e.g. AF as an abbreviation of applied force).

The content of the ith column of the metabase is of the following form:

$$ABR(var_i) - ABR(par_i) = p_i : i : ord_i$$

where $ABR(var_i)$ stands for the abbreviation of the ith variable located at the ith field of the metabase. "ord_i" is a symbol ($<, >, !$) to denote one storage-order of the numerical data. If a column can be characterized with a parameter, the notation $ABR(par_i) = p_i$ is used, where p_i denotes the value associated with that parameter. If not (i.e. n parameters are necessary), then the following form is used for the description of the ith column:

$$ABR(var_i) - ABR(par_i^1 = p_i^1 \ldots par_i^n = p_i^n) : i : ord_i.$$

Table 2. Part of the metabase with respect to laser-material interactions. The last line summarizes the storage-orders of the data of Table 1.

/acirect	acirpol			
#acipol	T-:1:>	AF-:2:>		
/au4gel	acipol			
/au4po	acipol			
#acipo	PD-:1:>	T-TH=1:2:<	FL-TH:1:3:!	
#carbres	AF-:1:<	T-PD=950:2:>	T-PD=500:3:>	T-PD=240:3:!

As seen in the next section, the necessity of using one or more parameters is dictated by the insufficiency of having only the name of the tables and the name of the current attribute(s) of a query when trying to identify the relevant databases.

The metabase can be viewed as a reservoir containing the storage orders of the data of those multi-databases having multi-facet-attributes. In general, knowing if a database has the above criteria is based on recognition of the identical attributes (i.e. substrings) that may exist between the names of the columns (i.e. variables). For instance, this happens among the second, third and fourth attributes of Table 1 due to $A = \frac{T}{PD}$. So, for those databases with multi-facet-attributes, the metabase described in this paper allows the heterogeneous databases to be transformed into "virtually" homogeneous ones, as appears in Table 2 which allows the direct addresses of the attributes to be gathered for efficient use later in the query processing. The reader familiar with join-processing algorithms would notice that having such above direct addresses allows one to reduce the time of join operations whereas, if one performs the join operations using a traditional approach, as mentioned in the previous section, that time increases.

ALGORITHM

In this section, first, the procedure of query-recognition is described, then, cooperation between the mentioned procedure and KBS for outputting the query-response is discussed.

Recognition of User-Query

Each long string of the query and the selector(s) is examined by way of the recognition of all keys of that long string. For faster processing of key-recognition, as well as associating a value for each recognized key, instead of using the conventional methods, such as hash-code techniques or transducer ones, the author's method was used, which was first sketched in [17], then described, along with algorithms, in [18]. This work has also been applied to other applications [19,20].

The essence of this method for faster processing of the keys and values associated with the keys, is as follows: Although finite-state transducers [21,22] can be used to map a language onto a set of values, the author's approach is based on an alternate representation method for such a mapping, consisting of associating a finite-state automaton (g) and accepting the input language (i.e. the set of the keys denoted by K) with a decision tree (dt)representing the output values. The advantages of this approach are that it leads to more compact representations than transducers and decision trees can easily be synthesized by machine learning techniques. The reader unfamiliar with automata theory and machine learning is referred to [23,24] and [25,26], respectively.

The recognition of the query is performed by the function "Query Recognition", which works as follows: Split each long string according to the space separator (or dash, if any) and collect the keys into a list of keys (lk). Spell out each key using g and this time, search its value, using dt.

If a key is unknown (i.e. $K \notin K$) and if it is an integer, then attach 'INT' as the abbreviation to the key, otherwise stop the processing. Collect the pair of (k, v) into a list, noted by lp, where v stands for an output value. Repeat the above step while adding new lp to the previous one. It may happen that due to some abbreviations (e.g. PD i.e. power density and T i.e. time), the gathering rules will be called upon. An example of such a rule is as follows: If k_1 = 'T' and k_2 = 'PD', then, form a new key (k), attach a new value to k and update lp.

```
func QueryRecognition()
    $lp \leftarrow \emptyset$; $nk \leftarrow 0$; $lspk \leftarrow \emptyset$;
    for each long string (ls) of the query do
      $lk \leftarrow Split\ (ls)$;
      for each k in lk do
         if SpellOut (k, g) then
            $v \leftarrow SearchValue\ (k, dt)$;
            Collect (k,v) into ls;
       else
         if IsSpecialKey (k) then
           Collect nk into lspk; else exit;
         end if
        end if
        nk++;
       end for
      end for
      if $lspk \neq \emptyset$ then
          ApplyGatheringRules();
    end if
cnuf
```

Cooperation with KBS

Two large bodies of declarative knowledge are present: Databases and background knowledge. Background knowledge contains domain specific information, used to supply qualitative information about the scope of the knowledge system. Background knowledge guides toward the proper files, whereas data knowledge leads to the proper fields, in a given file (database). Figure 2 illustrates an external object description of the quantitative (numerical) files. In order to allow properties to be inherited down from classes to subclasses and from classes to objects, NEXPERT's default strategy is used.

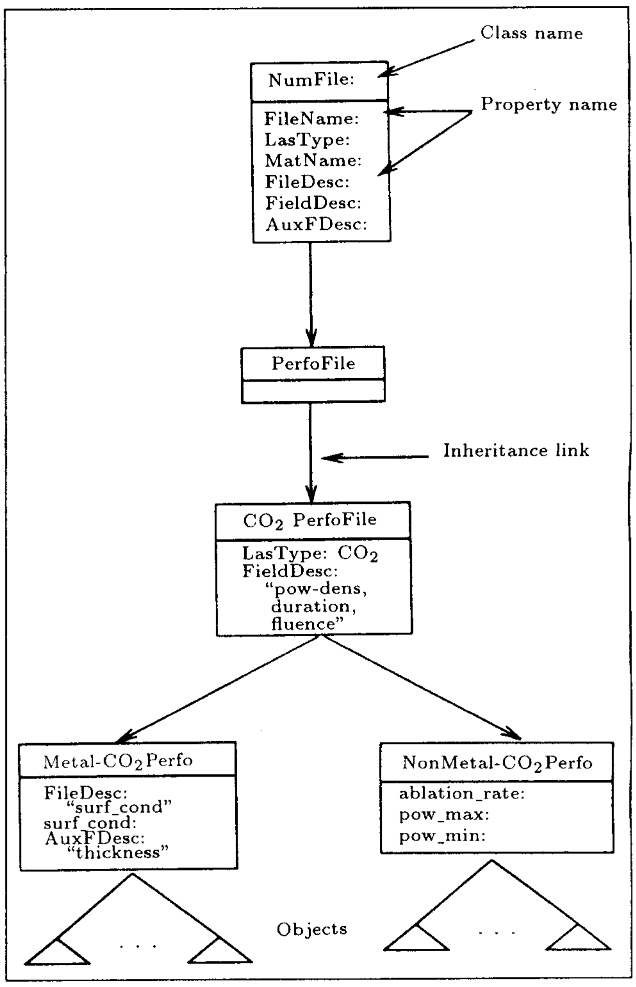

Figure 2. Part of quantitative data file representation.

The databases (quantitative files) contain experimental results describing the effects of a given irradiation on a given sample for the purpose of a given damage to the material. The material name and the laser type may be insufficient to characterize the file where the data are stored. In this case, some other parameters may be necessary to define completely which file contains the proper data. These parameters file descriptors will be noted, since they characterize a file. Material name, laser type and damage type are particular files. However, they stay out of the property file descriptor because they appear in all the files. Information concerning the damage type stays in the object structure as the class name of one of the parents of the object-file.

KBS is a set of rules acting through the object knowledge. The rules are grouped in islands, each of them having a particular purpose. The rules managing the retrieval process are split into four parts (see Figure 3).

The first part searches file names, which will be necessary (in part) to answer the user question. To reach this goal, if the second type of the acquisition of the queries (see the section of Architecture of Q2) is selected, then one is asked for the topic of his/her request and the field descriptor linked to this topic. If not, depending on the content of the queries, an exhaustive search algorithm, not described here, will be applied to find the topic's name. Then, pattern matching is done on the object representation of the files and the chosen files are linked to a common object.

The second rule set creates a parameter-object for each parameter of the studied domain. The parameter corresponding to the information requested by the user is assigned to the unknown class.

The third part is designed to instantiate all the objects created by the previous rule set. Known parameters are directly required from the user and assigned to the known class, whereas the unknown is found by means of the external loop through the encoder and accommodator (interpolator). Its value is put in the corresponding object.

The last part stores all information contained in the parameter-objects into objects, which are domain specific. Then, it checks whether some other files have to be investigated. If this is the case, the previous rule set is triggered again; otherwise an extern C program is called to display the results which have been computed.

Examples of Rules

The following examples show some NEXPERT-based rules. When all selected files are handled, the display module of the following rule creates the text-file, namely, dispfile, summarizing all the results. It will then be used to display the results.

```
If SelectedFiles.nbr is precisely equal to 1
Then display_results is confirmed.
And Execute "Display"(@ATOMID=<|Irradiation|>;@STRING =
"V(FileKey.unknown_value)";)
dispfile.txt"@KEEP=TRUE; @WAIT=FALSE;
```

The following rules illustrate the iterative process for storage of the values to be determined, using other rules (e.g. save, instantiation), as well as for displaying the results.

```
If SelectedFiles.nbr is greater than 1
And there is evidence of run
Then save is confirmed.

If SelectedFiles.nbr is greater than 1
And there is evidence of post_treatment
Then save is confirmed.

And Execute "GetRelatives"
     (@ATOMID=|KnownParam|;
     @STRING = "OBJECTS,@CHILDREN,
     @EVERYLEVEL,@RETURN=KnownParam.type";)
And Execute "GetRelatives"
     (@ATOMID=|UnknownParam|;
     @STRING = "OBJECTS,@CHILDREN,
     @EVERYLEVEL,@RETURN=UnknownParam.type";)
And Execute "Instantiation"
     (@ATOMID='File'_\FileKey.objFile\,
     |UnknownParam|.type,
     |UnknownParam|.type";)
And Reset run.
```

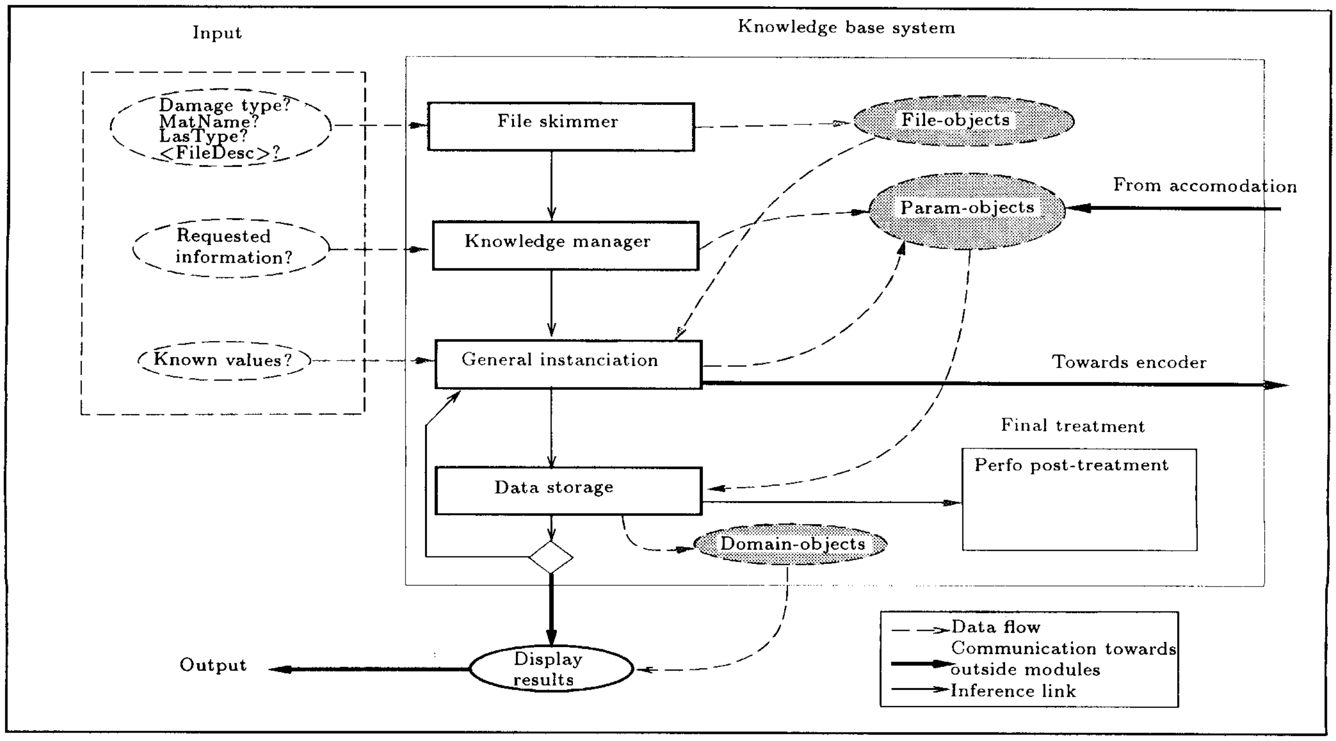

Figure 3. Reasoning Q2 overview.

```
If SelectedFiles.nbr is less than or equal than 1
And there is evidence of post_treatment
Then save is confirmed.
And Execute "GetRelatives"
    (@ATOMID=|KnownParam|;
    @STRING = "OBJECTS, @CHILDREN,
    @EVERYLEVEL,@RETURN=KnownParam.type";)
And Execute "GetRelatives"
    (@ATOMID=|UnknownParam|;
    @STRING = "OBJECTS,@CHILDREN,
    @EVERYLEVEL,@RETURN=UnknownParam.type";)
And Execute "Instantiation"
    (@ATOMID='File'_\FileKey.objFile\,
    |UnknownParam|. type,
    |UnknownParam|.type";)
```

The rule in charge of save (with high priority) first verifies if there are still one or more files to be inspected. If any, this rule calls the instantiation-rule with object-argument, wherein two distinct lists of known and unknown variables have been passed. Readers may refer to [27, pp 22-41] for a complete description of rules and their interactions.

ARCHITECTURE OF Q2

Q2 is designed to retrieve information and has been implemented using the C language and NEXPERT tool. That is to say, Q2 analyzes the input, stores the knowledge related to the input request, draws inferences from that knowledge, searches information related to the query and generates the answer. Q2 includes the following components: Knowledge Base System (KBS), lexicon, encoder, decoder, data retriever and several tabulated files. Figure 4 shows the schematic of Q2.

Acquisition of User-Queries

Two variants of the acquisition are available in Q2. The first one is designed for domain experts who are supposed to be familiar with background knowledge of the application at hand. For instance, a practitioner of laser-material interaction, when querying for determining the perforation time of a given steel, is supposed, not only to provide the name of the material, but, in general, its type, subtype and etc., such as steel35ncd13. Therefore, it is not so strange to expect receiving as the value of the templates of the query, the long strings, such as the following:

"Material name of the target = steel35cnd".

The second variant is conceived for non-experts and will be used below for illustration of an example of query processing. In this variant, KBS sends a list of topic-queries to the user. After having received the user-response, first, it searches and selects the list of topic-dependent questions (see Step 2 of the example given below). Second, according to the content of the user-response, KBS chooses the list of dependent-implicative questions and stores the user-response. Apart from the metabase and interpolator described, the short descriptions of other components of Q2 (i.e. decoder and encoder) are given below.

Decoder

It works on the lexicon, a simple database with at least two fields. The first field contains used KBS terms, whereas the second represents the term's short form (or abbreviation). This module decodes KBS terms in order to be usable by the encoder for being used by the

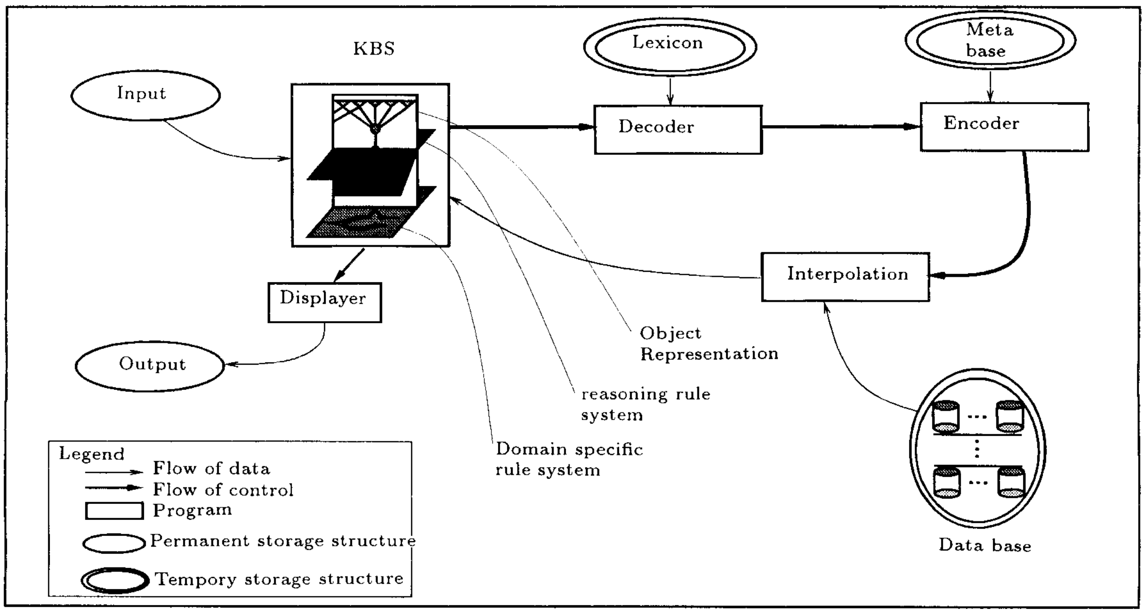

Figure 4. Architecture of Q2.

interpolation of selected plots. It is important to note that the lexicon can be located wherever a translation is necessary.

Encoder

The purpose of this module is to obtain the retrieval information for the interpolation. The Encoder module simplifies the decoder's result and uses the data retriever file for retrieval purposes. First, the encoder module gets the name of the file, second, by using the data retriever file it replaces the names of known and unknown user-question parameters by field numbers (see Step 6 of the following subsection of an example).

Finally, KBS receives the response given by the interpolation and transmits it to the user.

Example

By way of the second variant of the acquisition of user-queries described above, the implication of the presented architecture for the testing example will be illustrated step by step.

- How long does it take to perforate a 2 mm thick sheet of steel with a CO_2 laser irradiation of 3500W/cm^2?

Step 1: Topic Question Name

KBS interface displays the topic-question list and waits for the user to choose: Topic-Question Name? $\Rightarrow$ damage.

Note that the left and right side of $\Rightarrow$ stand for the message sent by Q2 and user-response, respectively.

Step 2: Dependent-Topic Question Answering

According to the previous response given by the user, there exist four dependent-topic questions, listed below:

Kind of damage under study? $\Rightarrow$ perforation
Material name of the target? $\Rightarrow$ steel35ncd13
Surface condition of the sample? $\Rightarrow$ UNKN.

Step 3: Identification of Reservoir Names

The later response, unknown, means that there exist several reservoirs (files), which must be inspected. By virtue of the three first user-responses: Perforation (class name), material and laser names, simple pattern matching is applied. The results are the filenames, namely, "PSS" (Polished Sheet of Steel) and "RSS" (Rectified Sheet of Steel).

Step 4: Implicative Questions

Due to the results in the third step, after having analyzed the responses, KBS selects a set of additional questions which are called the implicative questions.

Power density of the irradiation? $\Rightarrow$ 3500
Thickness of the target? $\Rightarrow$ 2 mm.

At the end of this step, two items of residual information are obtained, namely,

(RI1) : PSS, duration, pow_dens, 3500, 2 and
(RI2) : RSS, duration, pow_dens, 3500,2.

KBS instantiates by pattern-matching and by the laser name (CO_2) which uses the power density parameter.

Step 5: Short Form Given by Decoder

KBS will send to the decoder module the current residual information: "PSS, duration, pow_dens, 3500". The value associated to the thickness is two. Via its counterpart (the value associated with the thickness of 1 mm) and by the interpolator, the searched value is calculated. In this phase of processing, the lexicon will be called by the decoder in order to build a short form:

input: PSS, duration, pow_dens, 3500
output: RSS, DRT, PD, 3500.

Step 6: Retrieval Information Obtained by Encoder and Metabase

The encoder receives the results of the previous step, namely, "PSS, DRT, PD, 3500" and uses the metabase in order to find the appropriate retrieval information:

input: PSS, DRT, PD, 3500
output: PSS, 2, 1, $>$ 3500.

Recall that "PSS, 2, 1, $>$ 3500" means that the proper reservoir name is "pss.tab". The searched information must be retrieved by the second field and the known variable is in the first field. This field is arranged in an increasing order and the known variable value is 3500.

Step 7: Interpolation

If required, as is the case for the current example, this step will be solicited. For the thickness value (2 mm), the interpolator will give the searched time, say, $x1$, therefore:

input: 3500 and interpolator
output: $x1$ (via the interpolator given in the Figure 1.).

After passing from Step 4 to Step 7, the second residual information (i.e. R12) is considered and the values $x2$ are obtained. Finally, the responses to the user-question are:

For polished sheet of steel $= x1$
For rectified sheet of steel $= x2$.

CONCLUSIONS

In this paper, based on the notion of functional dependency, a plan to form the wrapper, with respect to the query processing, was devised for performing the analysis of the queries and then finding the searched information, including the implicit ones using other components of Q2.

The advantage of this work, compared to traditional query processing, can be advocated by a number of reasons, including: (1) Querying is user friendly, in the sense that it allows a free-syntax by just expressing the task by way of the long strings; (2) Compact representation (of the keys) and fast lookup is done by our own efficient method based on authomata theory and machine learning; (3) The implicit information are identified by wrapper and other component of Q2.

Although, it is believed that this present work has obvious advantages, with respect to both conventional query processing and mining tasks of query processing, however, a fair comparison (i.e. same data, same machine, etc.) is desired.

ACKNOWLEDGMENTS

This work was partially funded by the French-German Saint-Louis Institute. The help of Jean-Christophe Robert for the implementation of the knowledge base system part of this work is appreciated. Thanks also to the anonymous reviewers for giving hints to improve the presentation.

REFERENCES

1. Agrawal, R., Imelinski, T. and Swami., A. "Database mining: A performance perspective", *IEEE Transaction on Knowledge and Data Engineering*, **5**(6), pp 914-925 (1998).
2. Fayyad, G., Piatetskey-Sharpio, P. and Smyth, P. "From data mining to knowledge discovery: An overview", in *Advance in Knowledge Representation and Data Mining*, G. Fayyad, P. Platetskey-Sharplo, P. Smyth, and R. Uthurusamy, Eds., AAAI Press/The MIT Press, pp 495-515 (1966).
3. Han, J. and Fu, Y. "Attribute-oriented induction in data mining", in *Advance in Knowledge Representation and Data Mining*, G. Fayyad, P. Piatetskey-Sharpio, P. Smyth, and R. Uthurusaxny, Eds., AAAI Press/The MIT Press, pp 177-194 (1966).
4. Liu, H., Lu, H. and Yao, J. "Toward multi-database mining: Identifying relevant databases", *IEEE Transaction on Knowledge and Data Engineering*, **13**(4), pp 541-553 (July/Aug. 2001).
5. Zembowicz, R. and Zytkov, J.M. "From contingency tables to various forms of knowledge in databases", in *Advance in Knowledge Representation and Data Mining*, G. Fayyad, P. Piatetskey-Sharplo, P. Srnyth, and R. Uthurusaniy, Eds., AAAI Press/The MIT Press, pp 329-349 (1966).
6. Fatholahzadeh, A. "Querying by long strings in e-commerce", in *Collaborative Electronic Commerce Technology and Research*, M.A. Williirn and P. Swatman, Eds., Collecter, International Conference on Knowledge Representation, Toulouse, France, pp 1-6 (April 2002).
7. Henzinger, M. "Tutorial on web algorithms", in *Joint Sessions of FST, TCS and ISACCIAA* (Chennai, 1999).
8. Miller, R.C. and Bharat, K. "Sphinx: A framework for creating personal site-specific web crawlers", in *7th International WWW Conference*, Brisbane (1998).
9. Miller, R.C. and Bharat, K. "Querying the world wide web", in *Symposium on Parallel and Distributed Information Systems* (1996).
10. Pazzani, M., Muramatsu, J. and Billsus, D. "Syskill and webert: Identifying interesting web sites", in *13th American National Conference on AI (1996)*, Revised version in Machine Learning, **27** (1997).
11. Kushmerick, K. "Wrapper induction: Efficiency and expressiveness", *Artificial Intelligence*, **118**, pp 15-68 (2000).
12. Armstrong, W.W., *Dependency Structure of Database Relationship*, IFIP, North-Holland, Amesterdam (1974).
13. Maier, D., *The Theory of Relational Databases*, Computer Science Press, Rockville, MD (1983).
14. Mannila, H., and Räihä, K.-J., *Design of the Relational Databases*, Addison-Wesley, Wokingham (1992).
15. Mannila, H. and Räihä, K.J. "Algorithms for inferring functional dependencies", *IEEE Transaction on Knowledge and Data Engineering*, **12**, pp 83-99 (1994).
16. Ibaraki, T., Kogan, A. and Makino, K. "Functional dependency in horn theories", *Artificial Intelligence*, **108**, pp 1-30 (1999).
17. Fatholahzadeh, A. "DAWG-ID3- retrieving key-information using graph and classification algorithms", in *International Symposium on Database Technology & Software Engineering, WEB and Cooperative Systems*, G.E., Lasker and W., Gerhardt, Eds., Baden-Baden, Germany, pp 117-124 (Aug. 2000).
18. Fatholahzadeh, A. "Implementation of dictionaries via automata and decision trees", in *International Conference on Implementation and Application of Automata*, J. Champarnaud and D. Maruel, Eds., Tours, France, pp 101-110 (July 2002).
19. Fatholahzadeh, A. "Experiments with automata and information gain", in *International Conference on Implementation and Application of Automata*, M., Daley, M.G., Eramian, and S., Yu, Eds., Ontario, Canada, p 252 (July 2000).
20. Fatholahzadeh, A. "Nationality word graph for fast information retrieval", *Collaborative Electronic Commerce Technology and Research* (Colorado, USA, April 2000), M.A. Williams and P. Swatman, Eds., Collecter, *International Conference on Knowledge Representation*, pp 1-12, Available via www.collecter.org/collUSA.

21. Mohri, M. "Finite-state transducers in language and speech processing", *Computational Linguistics*, **23**(2), pp 269-311 (1997).

22. Mohri, M. "Generic ϵ-removal algorithm for weighted automata", in *International Conference on Implementation and Application of Automata*, M. Daley, M.G. Eramian and S. Yu, Eds., Ontario, Canada, pp 26-35 (July 2000).

23. Hopcroft, J.E. and Ullman, J.D., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, Massachusetts (1979).

24. Rozenberg, G., and Salomaa, A., Eds., *Handbook of Formal Language*, Springer-Verlag, Berlin Heidelberg (1997).

25. Mitchell, T., *Machine Learning*, McGraw-Hill (1997).

26. Quinlan, R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann (1993).

27. Robert, J.C. "Système d'aide à l'interpretation de résultats expérimentaux dans le domaine de l'interaction laser-matiére. Tech. rep.", Department of Computer Science, Supélec-Metz, 2, rue Édouard Belin 57070 Metz, France (June 1993).